HTML Lists

1. Types of Lists

Theory:

HTML provides three list types for different semantic purposes:

- **Unordered lists (`<ul>`)**: For collections where item order doesn't matter (e.g., features, options)
- **Ordered lists (`<ol>`)**: For sequences where order is significant (e.g., steps, rankings)
- **Description lists (`<dl>`)**: For term-definition pairs (e.g., glossaries, metadata)

Practical Syntax:

```html
<!-- Unordered -->
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>

<!-- Ordered -->
<ol type="I">
  <li>First step</li>
  <li>Second step</li>
</ol>

<!-- Description -->
<dl>
  <dt>Term</dt>
  <dd>Definition</dd>
</dl>
```

2. Block vs Inline Elements

Theory:

Block Elements	Inline Elements
`<div>` , `<p>`	`<span>` , `<a>`
`<h1>` - `<h6>`	`<strong>` , `<em>`
`<ul>` , `<table>`	`<img>` , `<input>`

- **Block elements** (e.g., `<div>` , `<p>` , `<ul>`):
  - Create "blocks" that stack vertically
  - Occupy full available width
  - Can contain other blocks/inline elements
- **Inline elements** (e.g., `<span>` , `<a>` , `<strong>`):
  - Flow within text/content
  - Only occupy necessary width
  - Cannot contain block elements

Key Differences:

1. Blocks stack vertically; inline flows horizontally
2. Blocks can contain inlines; inlines **cannot** contain blocks
3. Use CSS `display` to override (e.g., `span {display: block;}`)

3. The `<div>` Element

Theory:

- Generic block-level container
- No inherent semantic meaning
- Primary uses:
  - CSS styling/grouping
  - JavaScript DOM manipulation
  - Layout structure

Practical Example:

```html
<div class="card" id="user-profile">
  <h2>User Details</h2>
  <p>Email: user@example.com</p>
</div>
```

4. Classes vs IDs

Theory:

- **Classes**:
  - Reusable across multiple elements

    - CSS selector: `.classname`
    - JavaScript access: `getElementsByClassName()`
- **IDs**:
    - Unique per document
    - CSS selector: `#idname`
    - JavaScript access: `getElementById()`

When to Use:

```
- **Use classes** when:
  * Styling multiple elements similarly
  * Grouping related elements

- **Use IDs** when:
  * Targeting single unique elements
  * JavaScript needs direct element access
```

Practical Exercises

1. **Semantic Lists** Create a recipe page with:
    - Ordered list for steps
    - Unordered list for ingredients
    - Description list for nutrition facts
2. **Layout Construction** Build a 3-column layout using `<div>` blocks with:
    - Header (full width)
    - Main content + 2 sidebars
    - Footer (full width)
3. **CSS Targeting** Style elements using:
    - Class selectors for all buttons
    - ID selector for main navigation
4. **Element Nesting** Demonstrate proper nesting:
    - Block elements containing inline elements
    - Invalid nesting (and how to fix it)
5. **Accessibility** Enhance lists with:
    - ARIA roles for screen readers
    - Keyboard-navigable components

Key Principles:

1. Always prefer semantic HTML over generic containers when possible
2. Use lists for their intended semantic purpose, not just visual presentation
3. Reserve IDs for truly unique elements only
4. Understand the document flow differences between block and inline elements